Progetto SOASEC

Marco Pedrinazzi

Settembre 2022

1 Introduzione

Il progetto sviluppato ha l'obiettivo di dimostrare come è possibile proteggere delle *API REST* realizzate tramite *Node.js (framework Express)* con l'utilizzo di *Keycloak*. Sono state creati tre livelli di sicurezza per le API associati a due utenti di test: *pubblico* (nessuna sicurezza), *user* e *admin*. Le API di riferimento sono:

- \ (*homepage* della applicazione web) (visibilità pubblica)
- **\op1** (visibilità pubblica)
- **\users** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)
- **\users\op1** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)
- **\admin** (visibilità limitata agli utenti con il ruolo: *admin*)
- **\admin\op1** (visibilità limitata agli utenti con il ruolo: *admin*)

Il deployment della applicazione web costruita con Node.js (framework Express) e del server Keycloak è stato realizzato tramite *Heroku*, un famoso *PaaS* che consente agli sviluppatori di creare, eseguire e gestire applicazioni interamente nel *cloud*.

E' possibile visitare l'applicazione web all'indirizzo `https://marco-webapp.herokuapp.com` mentre il server Keycloak è disponibile all'indirizzo `https://marco-keycloak.herokuapp.com/auth/`. Inoltre, il codice sorgente della applicazione web è accessibile su *GitHub* all'indirizzo `https://github.com/marcopedrinazzi/esame-marco`.

Di seguito vengono dettagliate le scelte compiute in fase di implementazione insieme a maggiori dettagli sul progetto.

2 Implementazione

Le principali tecnologie utilizzate sono state Keycloak, Node.js (framework Express), *Docker* e Heroku.

2.1 Keycloak

Keycloak è una soluzione open source di *identity e access management*. Essa consente di aggiungere l'autenticazione alle applicazioni e proteggere i servizi con il minimo sforzo senza occuparsi della memorizzazione e dell'autenticazione degli utenti. [Key22a]

Keycloak supporta sia *OpenID Connect (OIDC)* che *SAML 2.0*. Visto la scelta di utilizzare Node.js, sarà necessario utilizzare il relativo *Keycloak client adapter* (una libreria che rende molto facile proteggere applicazioni e servizi Node.js con Keycloak) il quale è basato su OIDC. Quest'ultimo è un protocollo di autenticazione che è un'estensione di OAuth 2.0. OIDC fa un uso massiccio del set di standard *JSON Web Token (JWT)*. Questi standard definiscono un formato JSON del token di identità e le modalità per firmare digitalmente e crittografare tali dati in modo compatto e compatibile con il web. [Key22b]

Grazie alle funzionalità di Heroku, Keycloak è stato configurato per utilizzare come database *PostgreSQL*, una soluzione matura, completa ed ideale anche per uno scenario di produzione, invece del database *H2*, adatto solo ad uno scenario di test e sviluppo.

Si riportano di seguito i principali parametri di configurazione utilizzati nel server Keycloak, per ulteriori dettagli è possibile effettuare il log-in nella *Administration console* all'indirizzo `https://marco-keycloak.herokuapp.com/auth/`.

Creazione e configurazione del *realm* Un realm gestisce un insieme di utenti, credenziali, ruoli e gruppi. Un utente appartiene e accede a un realm. I realm sono isolati l'uno dall'altro e possono solo gestire e autenticare gli utenti che controllano. [Key22c] Il realm creato è stato chiamato *soasec* ed è stato impostato il requisito di SSL su tutte le richieste. Il resto delle impostazioni è stato lasciato uguale alle impostazioni di default.

Creazione e configurazione del *client* I client sono entità che possono richiedere Keycloak per autenticare un utente. Nella maggior parte dei casi, i client sono applicazioni e servizi che desiderano utilizzare Keycloak per proteggersi e fornire una soluzione *single sign-on*. I client possono anche essere entità che desiderano semplicemente richiedere informazioni sull'identità o un token di accesso in modo da poter invocare in modo sicuro altri servizi sulla rete protetti da Keycloak. [Key22c] Il client creato si chiama *my-secure-app*. La configurazione di quest'ultimo ha previsto la configurazione di vari parametri. Il *login theme* è stato impostato a *keycloak* per garantirne un migliore aspetto grafico. Il *client protocol* è stato impostato, come menzionato sopra, con *openid-connect*. L'*access type* è stato impostato a *confidential*, cioè imponendo la conoscenza di un segreto per iniziare il protocollo di login. E' stata impostato su *ON* il parametro *Service accounts enabled* per permettere al client di autenticarsi con Keycloak ed ottenere il rispettivo *access token*. E' stato impostato a *ON* anche il parametro *Authorization enabled* per garantire il supporto ad autorizzazioni *fine-grained*. Infine, sono stati impostati propriamente i parametri

di *Root URL*, *Base URL* e *Valid Redirect URL*. Il resto delle impostazioni è stato lasciato uguale alle impostazioni di default.

Creazione e configurazione dei ruoli I ruoli identificano un tipo o una categoria di utente. *Amministratore, utente, manager e dipendente* sono tutti ruoli tipici che possono esistere in un'organizzazione. Le applicazioni spesso assegnano l'accesso e le autorizzazioni a ruoli specifici piuttosto che a singoli utenti, poiché la gestione degli utenti può essere troppo fine e difficile da gestire. [Key22c] Sono stati creati due ruoli specifici per il client: *admin* il quale rappresenta un ruolo associato ad utenti con privilegi elevati e *user* il quale rappresenta un ruolo associato ad utenti con privilegi non elevati. In seguito sono stati creati due ruoli a livello di realm *app-admin* e *app-user* alla quale sono stati associati i ruoli del client, facendoli diventare dei ruoli composti. La differenza tra ruoli a livello di client e a livello di realm è legata al fatto che i primi sono ruoli specifici che possono essere utilizzati solo da tale client, mentre i secondi sono ruoli globali utilizzabili da qualunque client. La scelta di usarli entrambi è legata ad una maggiore flessibilità.

Creazione e configurazione degli utenti Gli utenti sono entità che sono in grado di accedere al sistema. Possono avere attributi associati a sè stessi come e-mail, nome utente, indirizzo, numero di telefono e giorno di nascita [Key22c]. Sono stati creati due utenti: *marco.pedrinazzi* (privilegi *app-admin*) e *luca.rossi* (privilegi *app-user*).

2.1.1 Keycloak Node.js adapter

Allo scopo di integrare il funzionamento di Keycloak con quello di Node.js è stato utilizzato il *Keycloak Node.js adapter*, il quale richiede una configurazione preliminare utilizzando dei parametri appositi di Keycloak. Il tutto è consultabile nel file *app.js* nella repository GitHub del progetto. In particolare, la *role-based authorization* alla base della protezione delle API viene realizzata con il codice seguente:

```
app.use('/users', keycloak.protect(['user','admin']),usersRouter);
app.use('/admin', keycloak.protect('admin'), adminRouter);
```

In maniera da proteggere tutte le API ciascuno dei router specificati con i ruoli adeguati. In fine, è stato configurato il logout invocato dall'utente al path */logoff*, il che eseguirà un *redirect* alla homepage della applicazione web.

2.2 Node.js (framework Express)

Node.js è un ambiente di *runtime* multipiattaforma *open source* che consente agli sviluppatori di creare tutti i tipi di strumenti e applicazioni lato server in JavaScript. Le capacità di Node.js sono però limitate. Se si desidera aggiungere una gestione specifica per diversi verbi *HTTP* (ad es. *GET, POST, DELETE*, ecc.), gestire separatamente le richieste a diversi percorsi, servire file statici o

utilizzare modelli per creare dinamicamente la risposta, è necessario utilizzare un framework web. Express è il framework web Node.js più popolare che fornisce un solido set di funzionalità per applicazioni web. [Doc22]

2.2.1 Routes

Il *routing* si riferisce al modo in cui un'applicazione risponde a una richiesta del client verso un determinato *endpoint*, che è un *URI* (o percorso) e un metodo di richiesta HTTP specifico (GET, POST e così via). Ogni rotta può avere una o più funzioni di gestione, che vengono eseguite quando la rotta viene abbinata.[Exp22a] Sono state create le seguenti rotte:

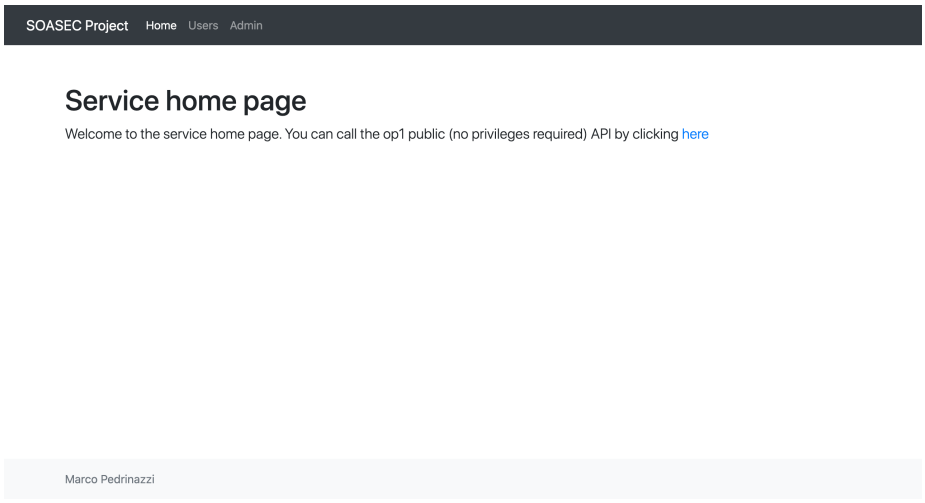
- \ (*homepage* della applicazione web) (visibilità pubblica)
- **\op1** (visibilità pubblica)
- **\users** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)
- **\users\op1** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)
- **\admin** (visibilità limitata agli utenti con il ruolo: *admin*)
- **\admin\op1** (visibilità limitata agli utenti con il ruolo: *admin*)

2.2.2 Views

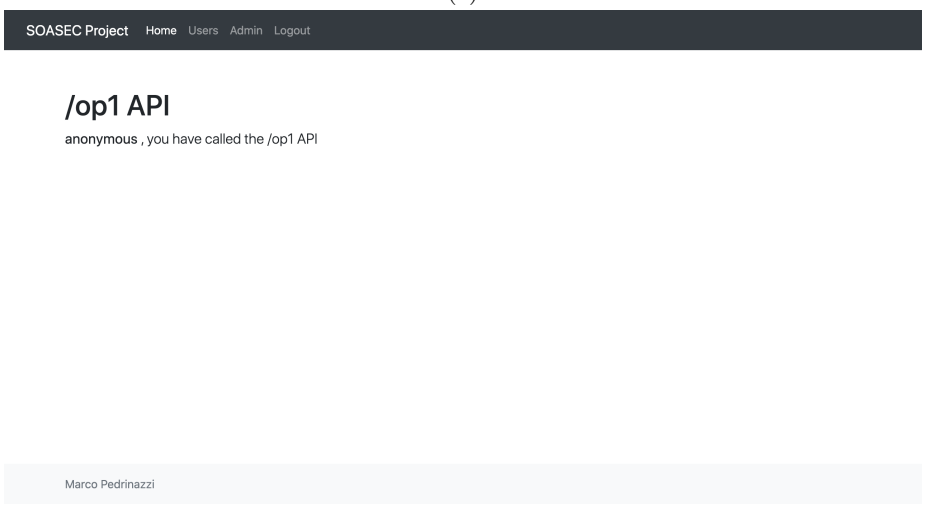
Un *template* engine consente di utilizzare file di template statici nell'applicazione. In fase di esecuzione, il template engine sostituisce le variabili in un file modello con i valori effettivi e trasforma il template in un file HTML inviato al client. Questo approccio semplifica la progettazione di una pagina HTML. [Exp22b]

Il template engine scelto è *EJS* considerate le sue proprietà di semplicità, efficacia e di supporto molto attivo. Il *front-end* della applicazione è realizzato con *Bootstrap*, il framework HTML, *CSS* e *JavaScript* più popolare per lo sviluppo di progetti *responsive* sul web.

Di seguito vengono riportate tutte le pagine che costituiscono l'applicazione web. Per maggiore dettagli sul *templating* e le viste si rimanda alla repository GitHub del progetto `https://github.com/marcopedrinazzi/esame-marco`.

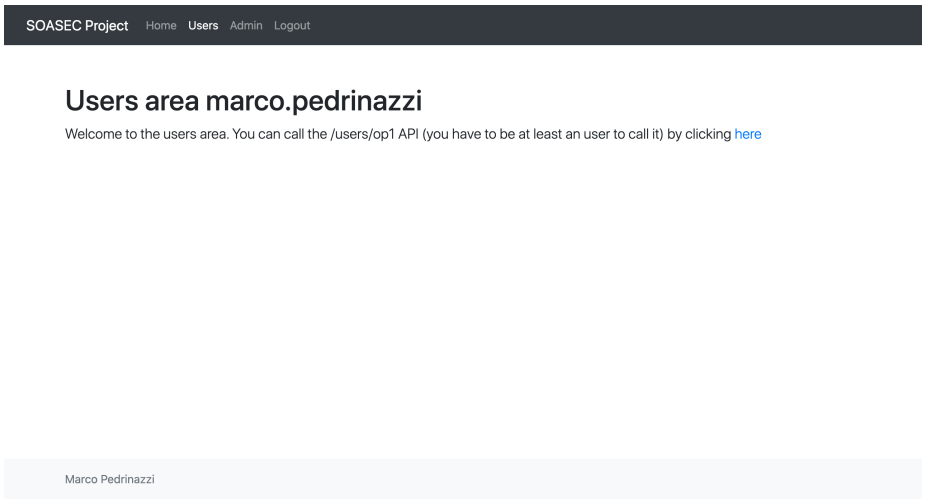


(a)

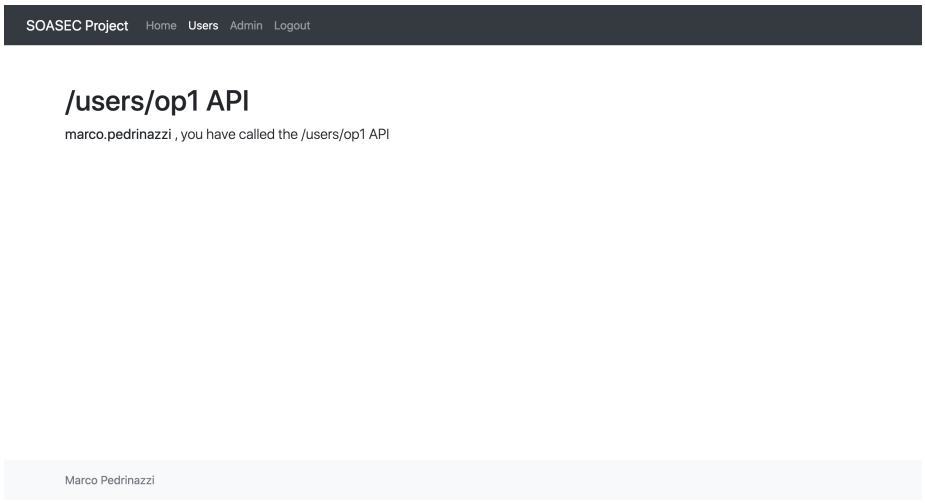


(b)

Figure 1: 1a) Home page della applicazione web (path \) (visibilità pubblica) 1b) Risultato dalla chiamata della API **\op1** (visibilità pubblica)

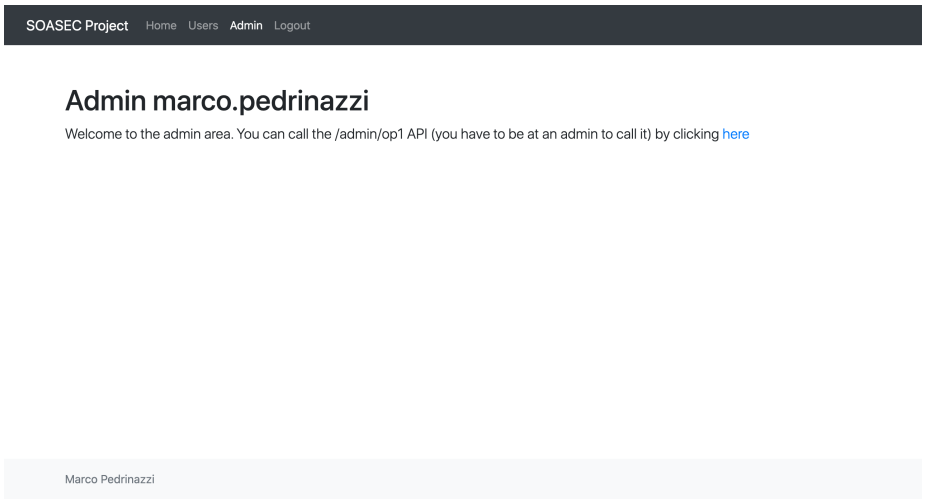


(a)

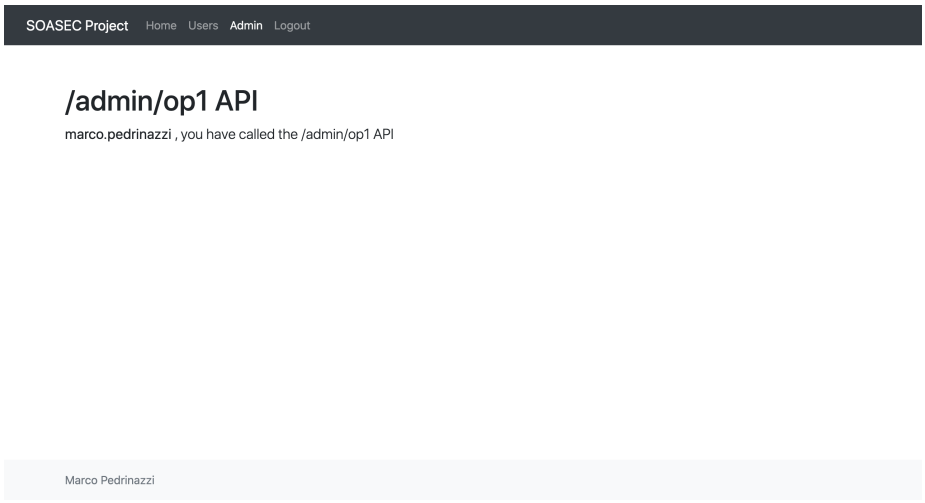


(b)

Figure 2: 2a) Risultato della chiamata della API **\users** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)
2b) Risultato della chiamata della API **\users\op1** (visibilità limitata agli utenti con i ruoli: *user* e/o *admin*)

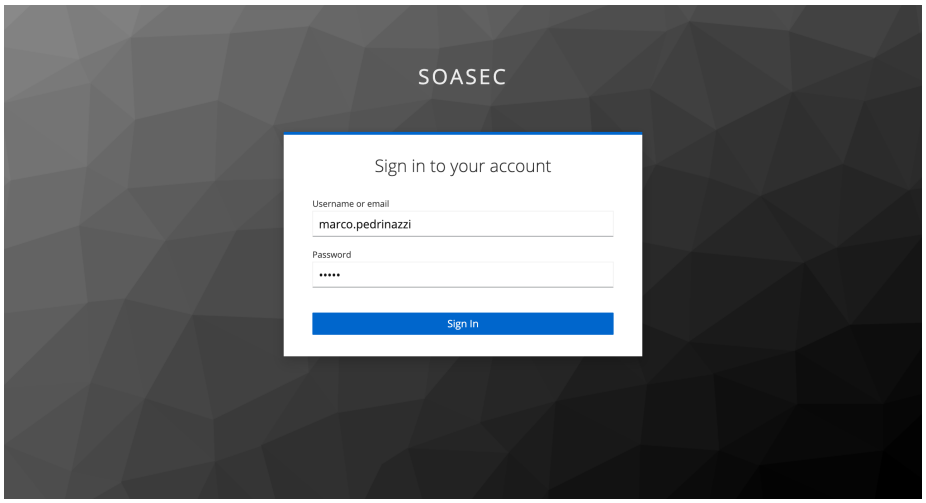


(a)

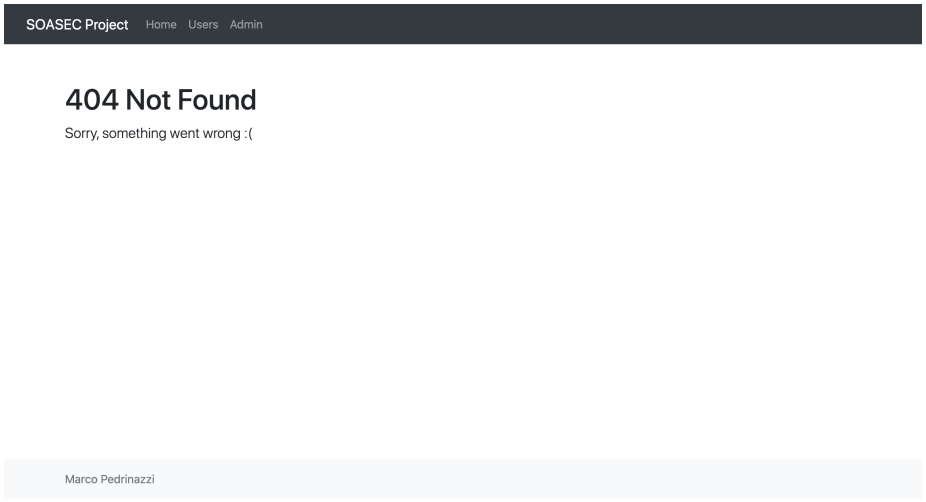


(b)

Figure 3: 3a) Risultato della chiamata della API **\admin** (visibilità limitata agli utenti con il ruolo: *admin*)
3b) Risultato della chiamata della API \admin\op1 (visibilità limitata agli utenti con il ruolo: *admin*)



(a)



(b)

Figure 4: 4a) **Schermata di sign in** che viene presentata ad ogni utente quando vuole accedere alle API Users/Admin.
4b) **Esempio di pagina di errore**. Questa viene modificata a seconda del tipo di errore che viene generato grazie ai meccanismi di templating.

2.3 Docker

Docker è una piattaforma open source che consente agli sviluppatori di creare, distribuire, eseguire, aggiornare e gestire *container*, componenti eseguibili standardizzati che combinano il codice sorgente dell'applicazione con le librerie del sistema operativo e le dipendenze necessarie per eseguire quel codice in qualsiasi ambiente. I container semplificano lo sviluppo e la distribuzione delle applicazioni distribuite. Sono diventati sempre più popolari man mano che le organizzazioni sono passate allo sviluppo cloud-native e agli ambienti multicloud ibridi. [IBM22]

Considerati i vantaggi di questa tecnologia, è stato effettuato il deployment del container per il server Keycloak su Heroku. (`https://github.com/sannonaragao/keycloak-heroku`)

2.4 Heroku

Heroku è un famoso PaaS che consente agli sviluppatori di creare, eseguire e gestire applicazioni interamente nel cloud. Heroku ha permesso di effettuare il deployment del progetto in una infrastruttura stabile, sicura, scalabile e dinamica. Infatti, sono state create due *Heroku app*, una per l'applicazione web (*marco-webapp*) e una per il deployment del container del server Heroku (*marco-keycloak*). L'app *marco-webapp* è collegata alla repository GitHub del progetto (`https://github.com/marcopedrinazzi/esame-marco`) ed effettua il deploy del *main branch* di quest'ultima. Viste le finalità del progetto, entrambe le applicazioni vengono eseguite su dei *dyno* gratuiti. I dyno sono container linux isolati e virtualizzati progettati per eseguire codice in base a un comando specificato dall'utente [Her22].

Come menzionato nella sezione 2.1, il deployment su Heroku del server Keycloak ha consentito di aggiungere gratuitamente alla sua configurazione un database PostgreSQL, già ideale per un contesto di produzione e non di solo sviluppo (per un'eventuale espansione futura del progetto).

L'utilizzo di Heroku ha consentito di garantire senza *(troppe)* configurazioni aggiuntive l'integrità e la confidenzialità di tutte le richieste da e verso l'applicazione web e il server Keycloak mediante l'utilizzo del protocollo *HTTPS* grazie ai certificati TLS offerti da Heroku.

References

[Doc22] MDN Web Docs. *Express/Node Introduction*. 2022. URL: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction` (visited on 09/07/2022).

[Exp22a] Express. *Basic routing*. 2022. URL: `http://expressjs.com/en/starter/basic-routing.html` (visited on 09/07/2022).

[Exp22b] Express. *Using template engines with Express*. 2022. URL: http://expressjs.com/en/guide/using-template-engines.html (visited on 09/07/2022).

[Her22] Heroku. *Heroku Dynos*. 2022. URL: https://www.heroku.com/dynos (visited on 09/09/2022).

[IBM22] IBM. *What is Docker?* 2022. URL: https://www.ibm.com/cloud/learn/docker (visited on 09/07/2022).

[Key22a] Keycloak. *Keycloak*. 2022. URL: https://www.keycloak.org (visited on 09/07/2022).

[Key22b] Keycloak. *Securing Applications and Services Guide*. 2022. URL: https://www.keycloak.org/docs/latest/securing_apps/ (visited on 09/10/2022).

[Key22c] Keycloak. *Server Administration Guide*. 2022. URL: https://www.keycloak.org/docs/latest/server_admin/ (visited on 09/09/2022).